Apple Assembly Line

$1.80

Volume 5 -- Issue 1 October, 1984

In This Issue...

Index to Volume 4

This time last year we published a cumulative index to the first three years of Apple Assembly Line. In this issue we add a separate index to Volume 4, covering October 83 through September 84. Perhaps in another year or two we can do another complete index.

65802 is Here!

After nearly a year of more or less patient waiting, we finally have a sample 65802 microprocessor. It does indeed plug right into an Apple //e, and works just fine. See Bob's story inside for all the details.

Blind Word Processor

Subscriber Larry Skutchan, of Little Rock, Arkansas, has adapted the S-C Word Processor to work with the Echo Two Speech Synthesizer. He now has a special word processor for the blind, which he says is the best available. The price will be $95.50. Larry is a blind university student, majoring in Computer Science. You can reach him at (501) 568-2172.

18-Digit Arithmetic, Part 6...............Bob Sander-Cederlof

This month's installment will cover some of the elementary functions: VAL, INT, ABS, SGN, and SQR. I will also introduce a general polynomial evaluator, which will be used by most of the other math functions.

Most of the functions expect a single argument, which will be loaded into DAC by the expression evaluator just before calling the function code. The function code will compute a value based on the argument, and leave the result in DAC. As the expression evaluator calls with JSR, the function code returns with RTS.

One exception to the above paragraph is the VAL function. VAL processes a string expression, and converts it into a value in DAC. The code in lines 1350-1610 of the listing closely parallels the VAL code in the Applesoft ROMs. Lines 1350-1370 evaluate the string expression. Lines 1380-1460 save the current TXTPTR value (which points into your Applesoft program), and makes TXTPTR point instead at the resulting string. Lines 1470-1520 save the byte just past the end of the string and store a 00 terminator byte in its place. FIN will evaluate the string, placing the numeric value into DAC. Then lines 1540-1600 restore the byte after the string and TXTPTR.

The INT function zeroes any digits after the decimal point in a number. A number in DAC has 20 digits. The exponent will be $00 if the value is zero, $01-40 if the value is all fractional, $41-53 if the value has from 1 to 19 digits before the decimal point, or $54-7F if the value has no fractional digits.

Lines 1650-1700 remove the $40 bias from the exponent. If the exponent was $00-40, DP.ZERO will force DAC to zero. Lines 1730-1740 check for the case of no fractional digits, and exit immediately. Lines 1750-1860 zero the digits after the decimal point. If the exponent was odd, there is one digit to be removed in the first byte to be cleared; the rest get both digits zeroed.

The simplest function is ABS, or absolute value. All it requires is forcing the sign positive, handled at lines 1910-1930.

Almost as simple is SGN, or sign function. SGN returns -1, 0, or +1, according as DAC was negative, zero, or greater-than-zero. Lines 1970-1980 check DAC.EXPONENT, which will be zero if-and-only-if DAC is zero. If the value is not zero, lines 1990-2030 force the value to be 1.0, while retaining the original sign.

SQR, the square root function, is more interesting. Do you remember the way you learned to take square roots in high school? Neither do I, but there is a handier way in computers anyway.

S-C Macro Assembler Version 1.0.....................................$80
S-C Macro Assembler Version 1.1...................................$92.50
Version 1.1 Update..$12.50
Source Code for Version 1.1 (on two disk sides)....................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code)..................$20
S-C Cross Reference Utility (with complete source code)..............$50
DISASM Dis-Assembler (RAK-Ware)....................................$30
Source Code for DISASM................................additional $30

S-C Word Processor (with complete source code)......................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15

(All source code is formatted for S-C Macro Assembler Version 1.1. Other assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.......................................each $15
Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.

QD#1: Oct-Dec 1980	QD#2: Jan-Mar 1981	QD#3: Apr-Jun 1981
QD#4: Jul-Sep 1981	QD#5: Oct-Dec 1981	QD#6: Jan-Mar 1982
QD#7: Apr-Jun 1982	QD#8: Jul-Sep 1982	QD#9: Oct-Dec 1982
QD#10: Jan-Mar 1983	QD#11: Apr-Jun 1983	QD#12: Jul-Sep 1983
QD#13: Oct-Dec 1983	QD#14: Jan-Mar 1984	QD#15: Apr-Jun 1984
QD#16: Jul-Sep 1984		

AWIIe Toolkit (Don Lancaster, Synergetics)............................$39
Quick-Trace (Anthro-Digital)..........................(reg. $50) $45
Visible Computer: 6502 (Software Masters)...............(reg. $50) $45
ES-CAPE: Extended S-C Applesoft Program Editor........................$60
Amper-Magic (Anthro-Digital)..........................(reg. $50) $45
Amper-Magic Volume 2 (Anthro-Digital).....................(reg. $30) $25
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95) $36

Blank Diskettes (Verbatim)............2.25 each, or package of 20 for $40
(Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
or $25 per 100
These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers........................... 6 cents each
QuikLoader EPROM System (SCRG)...........................($179) $170
D MAnual Controller (SCRG)...................................($90) $85
Switch-a-Slot (SCRG)...($190) $175
Extend-a-Slot (SCRG)..($35) $32

Books, Books, Books..........................compare our discount prices!
"Apple][Circuit Description", Gayler..................($22.95) $21
"Understanding the Apple II", Sather....................($22.95) $21
"Enhancing Your Apple II, vol. 1", Lancaster............($15.95) $15
Second edition, with //e information.
"Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95) $20
"Incredible Secret Money Machine", Lancaster............($7.95) $7
"Beneath Apple DOS", Worth & Lechner....................($19.95) $18
"Beneath Apple ProDOS", Worth & Lechner.................($19.95) $18
"What's Where in the Apple", Second Edition.............($19.95) $19
"6502 Assembly Language Programming", Leventhal.........($18.95) $18
"6502 Subroutines", Leventhal...........................($18.95) $18
"Real Time Programming -- Neglected Topics", Foster......($9.95) $9
"Microcomputer Graphics", Myers.........................($12.95) $12

We have small quantities of other great books, call for titles & prices.
Add $1.50 per book for US shipping. Foreign orders add postage needed.

Texas residents please add 6 1/8 % sales tax to all orders.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We accept Master Card, VISA and American Express ***

Suppose I want to find square root of 25. I could start with a wild guess, check it to see if I am close by squaring and comparing with 25, and then refining my guess until it is as accurate as I need. Suppose my wild guess is 7 (pretty wild!).

7*7 is 49, which is bigger than 25, so my next guess should be less than 7. Instead of just guessing wildly for the next one, why not take the average between 7 and 25/7? That average is 5.286. The average of 5.286 and 25/5.286 is 5.0076. The next one is 5.0000079. You can see that I am rapidly approaching the answer of 5.0.

The method of refining an approximation as exemplified above was derived originally be Sir Isaac Newton. His method involves calculus, can get quite complex, and applies to all sorts of problems. But in the case of the square root, it is as simple as averaging an approximation with the argument divided by the approximation.

It turns out that it is a very good method, because if you can get an initial approximation that has the first few digits right, the number of digits that are correct will slightly more than double each time you run through Newton's improver.

The next trick is to reduce the range of possible arguments from the full range of zero to 10^63 down to the range from .1 to 1.0. The zero case is easy, because SQR(0) = 0, and is handled at lines 2100-2110. Notice that lines 2120-2130 weed out negative arguments, which are not allowed.

Remember that the square root of X*10^n is equal to SQR(X)*10^(n/2). Lines 2150-2190 save the exponent, and change it to $40. This changes the value in DAC to the range .1 to 1.0. I have a book which gives polynomial approximations to the square root in that range. One with the form aX^4+bX^3+...+e gives an approximation with is accurate in the first 2.56 digits. Three iterations by Newton yield more than 22 accurate digits. The same book shows a cubic polynomial which gives 2.98 accurate digits if we can get the value into the range between .25 and 1.0.

Lines 2200-2280 fold the values between .1 and .25 up to the range .4 through 1.0 by multiplying the value by 4. (This multiplication goes pretty fast, since most of the bytes are zero.) The fact that we quadrupled the value is remembered, so that we can later halve the approximate root at lines 2350-2410. The cubic polynomial is evaluated in lines 2290-2340, by calling POLY.N. The result, by the time we reach line 2420, is an approximate square root of the number between .1 and 1; now we need to make it an approximate root of the original argument.

Lines 2420-2480 compute the exponent of the square root, by simply dividing the original exponent by two. If there is a remainder, meaning the original exponent was odd, then we also need to multply the trial root by SQR(10). This is handled in lines 2490-2550. The halved original exponent next is added to the trial exponent, giving a good first approximation to the

square root of the original argument. Lines 2600-2740 run through three cycles of the Newton iteration, giving plenty of precision. If we were carrying enough digits along, the 2.98 digits of precision our polynomial produced would be refined to a full 26 digits, according to my book.

Speaking of the book, it is one I bought a number of years ago when working on double precision math for a Control Data 3300 time sharing system. As far as I know, it is still the best book in its field. "Computer Approximations", by J. F. Hart and about seven other authors, was published in 1968 by John Wiley & Sons. I don't know if it is still in print or not, but if you ever need to create some high precision math routines, you ought to try to find a copy.

A very common element in the evaluation of many math functions is an approximation to the function over a limited range by a polynomial, or by the quotient of two polynomials. Therefore it is handy to have an efficient subroutine to evaluate a polynomial. Two different entry points allow efficient evaluation of two kinds: those whose first coefficient is 1, and the rest. POLY.N evaluates those whose first coefficient is not one, and POLY.1 does those whose first is 1.

```
       POLY.N --  a*x^n + b*x^n-1 + ...
       POLY.1 --    x^n + a*x^n-1 + ...
```

In both cases, you enter with the address of a table of coefficients in the Y- and A-registers (hi-byte in Y, lo-byte in A), and the degree of the polynomial in the X-register. Thus you see that in lines 2290-2340 the table P.SQR is addressed, and the degree of polynomial is 3 (cubic). Both POLY.N and POLY.1 assume that the value of x is in TEMP2. Where all terms have been computed and added, the result will be in DAC.

Actually, I may have misled you a little in the last sentence. The terms of the polynomial are not separately computed and added, but rather they are accumulated in a simple serial fashion:

```
       poly = ((( a * x + b) * x + c) * x + d) * x + e
```

The coefficients and other constants shown in lines 2770-2830 are in a special format which includes an extra two digits. You will remember that the basic operations (+-*/) are carried out to 20 digits. Therefore these constants are carried out to 20 digits. They are not critical in the square root computation, thanks to Sir Isaac, but the log and trig functions will need them.

```
                    *SAVE S.DP18 FUNC 1
                    *--------------------------------
B7-                 AS.CHRGOT   .EQ $00B7
DD7B-               AS.FRMEVL   .EQ $DD7B
DD7B-               AS.CHKSTR   .EQ $DD7B
E600-               AS.FRESTR   .EQ $E600
E199-               AS.ILLERR   .EQ $E199
                    *--------------------------------
FFFF-               DMULT       .EQ $FFFF
FFFF-               DDIV        .EQ $FFFF
FFFF-               DADD        .EQ $FFFF
FFFF-               FIN         .EQ $FFFF
FFFF-               DP.TRUE     .EQ $FFFF
FFFF-               DP.ZERO     .EQ $FFFF
FFFF-               MOVE.DAC.TEMP3 .EQ $FFFF
FFFF-               MOVE.DAC.TEMP2 .EQ $FFFF
FFFF-               MOVE.TEMP2.DAC .EQ $FFFF
FFFF-               MOVE.YA.DAC.1  .EQ $FFFF
FFFF-               MOVE.YA.ARG.1  .EQ $FFFF
FFFF-               MOVE.TEMP3.ARG .EQ $FFFF
FFFF-               MOVE.TEMP2.ARG .EQ $FFFF
                    *--------------------------------
B8-                 TXTPTR      .EQ $B8,B9
60-                 DEST        .EQ $60,61
                    *--------------------------------
0800-               TEMP2       .BS 1
0801-               TEMP3       .BS 1
0802-               P1          .BS 2
0804-               DAC.EXPONENT .BS 1
0805-               DAC.HI       .BS 10
080F-               DAC.SIGN     .BS 1
                    *--------------------------------
                    *       VAL (X$) FUNCTION
                    *--------------------------------
0810- 20 B7 00      DP.VAL JSR AS.CHRGOT
0813- 20 7B DD             JSR AS.FRMEVL     GET STRING
0816- 20 7B DD             JSR AS.CHKSTR MAKE SURE IT IS A STRING
0819- A5 B8                LDA TXTPTR   SAVE TXTPTR
081B- 48                   PHA          ...ON STACK
081C- A5 B9                LDA TXTPTR+1
081E- 48                   PHA
081F- 20 00 E6             JSR AS.FRESTR FREE THE STRING;GET ADR IN
0822- 86 B8                STX TXTPTR   Y,X AND LEN IN A
0824- 86 60                STX DEST    SAVE BEGINNING OF STRING
0826- 84 B9                STY TXTPTR+1
0828- 84 61                STY DEST+1
082A- A8                   TAY          LENGTH TO Y
082B- 8D 00 08             STA TEMP2    SAVE LENGTH
082E- B1 B8                LDA (TXTPTR),Y
0830- 48                   PHA          SAVE CHAR AT END OF STRING
0831- A9 00                LDA #0
0833- 91 B8                STA (TXTPTR),Y  PUT 0 AT END OF STRING
0835- 20 FF FF             JSR FIN      GET THE NUMBER
0838- 68                   PLA          GET CHAR
0839- AC 00 08             LDY TEMP2    GET LENGTH
083C- 91 60                STA (DEST),Y
083E- 68                   PLA           RESTORE TXTPTR
083F- 85 B9                STA TXTPTR+1
0841- 68                   PLA
0842- 85 B8                STA TXTPTR
0844- 60                   RTS          VAL IS IN DAC
                    *--------------------------------
                    *       INT FUNCTION
                    *--------------------------------
0845- AD 04 08      DP.INT LDA DAC.EXPONENT
0848- 38                   SEC
0849- E9 40                SBC #$40     REMOVE OFFSET
084B- 10 03                BPL .1       POSITIVE EXP
                    *---ALL FRACTION, MAKE = 0-------
084D- 4C FF FF      .0     JMP DP.ZERO
                    *---SOME INTEGER, TRUNCATE-------
0850- F0 FB         .1     BEQ .0       ...ALL FRACTION
0852- C9 14                CMP #20      ALL INTEGER?
0854- B0 18                BCS .4       ...YES, NONTHING TO LOP
0856- 4A                   LSR          DIVIDE BY 2
0857- A8                   TAY          BYTE INDEX
0858- 90 0D                BCC .3       ...NO NYBBLE TO CLEAR
085A- B9 05 08             LDA DAC.HI,Y ...CLEAR A NYBBLE
```

```
085D- 29 F0     1780        AND #$F0
085F- 99 05 08  1790        STA DAC.HI,Y
0862- C8        1800 .2     INY            ...NEXT BYTE
0863- C0 0A     1810        CPY #10        FINISHED?
0865- B0 07     1820        BCS .4         ...YES
0867- A9 00     1830 .3     LDA #0         CLEAR A BYTE
0869- 99 05 08  1840        STA DAC.HI,Y
086C- F0 F4     1850        BEQ .2         ...ALWAYS
086E- 60        1860 .4     RTS
                1870 *--------------------------------
                1880 *      ABS (DAC)
                1890 *--------------------------------
086F- A9 00     1900 DP.ABS LDA #0         STORE 0 IN
0871- 8D 0F 08  1910        STA DAC.SIGN SIGN
0874- 60        1920        RTS
                1930 *--------------------------------
                1940 *      SGN (DAC)
                1950 *--------------------------------
0875- AD 04 08  1960 DP.SGN LDA DAC.EXPONENT
0878- F0 0B     1970        BEQ .1         IT IS 0, SO LEAVE IT
087A- AD 0F 08  1980        LDA DAC.SIGN
087D- 48        1990        PHA            SAVE SIGN
087E- 20 FF FF  2000        JSR DP.TRUE    PUT 1 IN DAC
0881- 68        2010        PLA
0882- 8D 0F 08  2020        STA DAC.SIGN RESTORE SIGN
0885- 60        2030 .1     RTS
                2040 *--------------------------------
                2050 *      SQR (DAC)
                2060 *      #0072 IN HART, ET AL
                2070 *--------------------------------
0886- 4C 99 E1  2080 ERR.SQ JMP AS.ILLERR  ILLEGAL QUANTITY
0889- AD 04 08  2090 DP.SQR LDA DAC.EXPONENT
088C- F0 7B     2100        BEQ .3           SQR(0)=0
088E- AD 0F 08  2110        LDA DAC.SIGN
0891- 30 F3     2120        BMI ERR.SQ     MUST BE POSITIVE
0893- 20 FF FF  2130        JSR MOVE.DAC.TEMP3 SAVE X
                2140 *---REDUCE RANGE TO .1 - 1-------
0896- AD 04 08  2150        LDA DAC.EXPONENT
0899- 48        2160        PHA            SAVE EXPONENT
089A- A9 40     2170        LDA #$40       CHANGE RANGE TO .1 THRU .9999...9
089C- 8D 04 08  2180        STA DAC.EXPONENT
                2190 *---REDUCE RANGE TO .25 - 1------
089F- AD 05 08  2200        LDA DAC.HI
08A2- C9 25     2210        CMP #$25       LESS THAN .25?
08A4- 08        2220        PHP            SAVE ANSWER
08A5- B0 0A     2230        BCS .4         ...NO
08A7- A9 4C     2240        LDA #CON.FOUR
08A9- A0 09     2250        LDY /CON.FOUR
08AB- 20 FF FF  2260        JSR MOVE.YA.ARG.1
08AE- 20 FF FF  2270        JSR DMULT
                2280 *---CALC FIRST APPROX.-----------
08B1- 20 FF FF  2290 .4     JSR MOVE.DAC.TEMP2
08B4- A9 0A     2300        LDA #P.SQR     GET FIRST APPROXIMATION
08B6- A0 09     2310        LDY /P.SQR     FROM AX^3+BX^2+CX+D
08B8- A2 03     2320        LDX #P.SQR.N
08BA- 20 75 09  2330        JSR POLY.N
                2340 *---ADJUST APPROX FOR FOLDING----
08BD- 28        2350        PLP            WAS X<.25?
08BE- B0 0A     2360        BCS .5         ...NO
08C0- A9 41     2370        LDA #CON.HALF
08C2- A0 09     2380        LDY /CON.HALF
08C4- 20 FF FF  2390        JSR MOVE.YA.ARG.1
08C7- 20 FF FF  2400        JSR DMULT
                2410 *---COMPUTE SQR EXPONENT---------
08CA- 68        2420 .5     PLA            GET EXPONENT FROM BEGINNING
08CB- 38        2430        SEC
08CC- E9 40     2440        SBC #$40       REMOVE OFFSET
08CE- 6A        2450        ROR            DIVIDE BY TWO (KEEP SIGN)
08CF- 49 80     2460        EOR #$80
08D1- 90 0C     2470        BCC .1         DON'T MULT BY SQR(10)
                2480 *---ADJUST APPROX FOR ODD EXP----
08D3- 48        2490        PHA            SAVE EXPONENT/2
08D4- A9 36     2500        LDA #CON.SQR10
08D6- A0 09     2510        LDY /CON.SQR10
08D8- 20 FF FF  2520        JSR MOVE.YA.ARG.1
08DB- 20 FF FF  2530        JSR DMULT
08DE- 68        2540        PLA
```

```
                  2550 *---INSTALL NEW EXPONENT---------
08DF- 18          2560 .1     CLC
08E0- 6D 04 08    2570        ADC DAC.EXPONENT
08E3- 8D 04 08    2580        STA DAC.EXPONENT
                  2590 *---THREE NEWTON ITERATIONS------
08E6- A9 03       2600        LDA #3
08E8- 8D 01 08    2610        STA TEMP3
08EB- 20 FF FF    2620 .2     JSR MOVE.DAC.TEMP2     TEMP2 = Y
08EE- 20 FF FF    2630        JSR MOVE.TEMP3.ARG     GET X
08F1- 20 FF FF    2640        JSR DDIV               X/Y
08F4- 20 FF FF    2650        JSR MOVE.TEMP2.ARG
08F7- 20 FF FF    2660        JSR DADD               X/Y+Y
08FA- A9 41       2670        LDA #CON.HALF
08FC- A0 09       2680        LDY /CON.HALF
08FE- 20 FF FF    2690        JSR MOVE.YA.ARG.1
0901- 20 FF FF    2700        JSR DMULT              (X/Y+Y)/2
0904- CE 01 08    2710        DEC TEMP3              ANY MORE?
0907- D0 E2       2720        BNE .2                 ...YES
0909- 60          2730 .3     RTS                    ...DONE
                  2740 *--------------------------------
03-               2750 P.SQR.N  .EQ 3
090A- 40 28 73
090D- 69 82 40
0910- 00 00 00
0913- 00 00       2760 P.SQR      .HS 4028736982400000000000
0915- C0 82 58
0918- 88 89 10
091B- 00 00 00
091E- 00 00       2770            .HS C082588889100000000000
0920- 41 13 22
0923- 56 38 60
0926- 00 00 00
0929- 00 00       2780            .HS 4113225638600000000000
092B- 40 21 70
092E- 18 67 20
0931- 00 00 00
0934- 00 00       2790            .HS 4021701867200000000000
0936- 41 31 62
0939- 27 76 60
093C- 16 83 79
093F- 33 20       2800 CON.SQR10  .HS 4131622776601683793320
0941- 40 50 00
0944- 00 00 00
0947- 00 00 00
094A- 00 00       2810 CON.HALF   .HS 4050000000000000000000
094C- 41 40 00
094F- 00 00 00
0952- 00 00 00
0955- 00 00       2820 CON.FOUR   .HS 4140000000000000000000
                  2830 *--------------------------------
                  2840 *      POLYNOMIAL EVALUATOR ROUTINES
                  2850 *      (Y,A) = ADDRESS OF COEFFICIENT TABLE
                  2860 *      ARRANGED HIGHEST POWER TO LOWEST
                  2870 *      CONSTANTS DO USE GUARD BYTE (11 TOTAL)
                  2880 *--------------------------------
                  2890 *   DO A POLYNOMIAL WITH 1ST CONSTANT 1
                  2900 *      (TEMP2) IS X-VALUE
                  2910 *      (X-REG) IS N
                  2920 *          WHERE N = POWER OF X
                  2930 *          FOR EXAMPLE, IF N=2 : X^2+AX+B
                  2940 *                          N=4 : X^4+AX^3+BX^2+CX+D
                  2950 *--------------------------------
                  2960 POLY.1
0957- 8D 02 08    2970        STA P1
095A- 8C 03 08    2980        STY P1+1
095D- 8E 01 08    2990        STX TEMP3
0960- 20 FF FF    3000        JSR MOVE.TEMP2.DAC
0963- AD 02 08    3010 POLY   LDA P1
0966- AC 03 08    3020        LDY P1+1
0969- 20 FF FF    3030        JSR MOVE.YA.ARG.1
096C- 20 FF FF    3040        JSR DADD
096F- CE 01 08    3050        DEC TEMP3     FINISHED YET?
0972- D0 0D       3060        BNE POLY2     ...NO
0974- 60          3070        RTS           ...YES
```

```
                   *-------------------------------
                   *      DO A POLYNOMIAL WITH 1ST CONSTANT <> 1
                   *      (TEMP2) IS X-VALUE
                   *      (X-REG) IS N
                   *          WHERE N = POWER OF X
                   *          FOR EXAMPLE, IF N=2 : AX^2+BX+C
                   *                          N=3 : AX^3+BX^2+CX+D
                   *-------------------------------
                   POLY.N
0975- 8D 02 08            STA P1
0978- 8C 03 08            STY P1+1
097B- 8E 01 08            STX TEMP3
097E- 20 FF FF            JSR MOVE.YA.DAC.1
0981- 20 FF FF     POLY2  JSR MOVE.TEMP2.ARG
0984- 20 FF FF            JSR DMULT
0987- 18                  CLC
0988- AD 02 08            LDA P1
098B- 69 0B               ADC #11       NUMBER OF BYTES
098D- 8D 02 08            STA P1
0990- 90 D1               BCC POLY
0992- EE 03 08            INC P1+1
0995- D0 CC               BNE POLY      ...ALWAYS
                   *-------------------------------
```

An Even Trickier "Index to Mask"...............Charles Putney
Dublin, Eire

I got AAL today (September 1984 issue), and pored through it as usual. The "index" article on page 18 caught my eye. Naturally I tried to think of a smaller way of coding the routine like your "TRICKIER.WAY" of 32 bytes. Here it is, in only 23 bytes!

```
*-------------------------------
*       PUTNEY'S WAY
*-------------------------------
PUTNEY.WAY
        AND #7          .....421
        LSR             ......42, 1 IN CARRY
        PHA             SAVE FOR LATER PLP
        LDA #1          INITIAL MASK VALUE
        BCC .1          NO NEED TO SHIFT 1
        ASL
.1      PLP             GET 1.....42 AS NV.BDIZC
        BCC .2          NO NEED TO SHIFT 2
        PHP
        ASL
        ASL
        PLP
.2      BNE .3          NO NEED TO SHIFT 4
        ASL
        ASL
        ASL
        ASL
.3      RTS
```

The timing, not including a JSR to it nor the RTS at the end, varies from a best case of 21 cycles to a worst case of 39 cycles.

[One note of warning: the PLP pulls a status of 000000xx, setting the I-status to zero. This enables IRQ interrupts, which might be very dangerous if you have an interrupting source connected and were otherwise unprepared.]

Correction to DP18, Part 5.......................Paul Schlyter

The following comments relate to the listing on page 13 of the September 1984 issue.

It appears to me that lines 4610-4620 and 4650-4660 can be deleted. They check for the non-tokenized forms of "+" and "-", which I believe will never be presented to DP18.

There is a definite bug at line 4460: the "LDA #$02" should be "LDA #$04". Compare with lines 4370 and 4410, and you will see how I caught it. Also the comment on line 4170, which says the bit map is in the form "00000<=>".

Another Tricky Way....Bruce Love
Hamilton, New Zealand

Here is my effort to improve your version of turning an index into a mask. It uses (shudder!) self-modifying code, but it is shorter and faster and I think easy to understand.

```
LOVE.VERSION
       AND #7
       EOR #7
       STA .1+1
       LDA #1
.1     BNE .1  (OFFSET FILLED IN)
       ASL
       ASL
       ASL
       ASL
       ASL
       ASL
       ASL
       RTS
```

And Still Another...David Eisler
Littleton, Colorado

With reference to "Turn Index into a Mask" (AAL Sept 84), here is another tricky alternative. It uses only the A-register, is only 16 bytes long, and takes 9 to 23 cycles.

```
EISLER.VERSION
       AND #7
       STA .1+1
       LDA #$80
.1     BNE .1  (OFFSET FILLED IN)
       LSR
       LSR
       LSR
       LSR
       LSR
       LSR
       LSR
       RTS
```

The 65802 is Here!.........................Bob Sander-Cederlof

I think it was last December that I learned of the new 16-bit versions of our old friend, the 6502. You will remember my enthusiastic description in the Jan 84 issue. People at Western Design Center were optimistic about shipping chips in a month or so. Very optimistic. Way too optimistic. Nevertheless, they followed the tradition of our whole industry by continuing to stick by their commitment. Every time we called, it was always "in a month or so"!

But yesterday (Oct 12th) it arrived. Nice shiny new COD sticker on top, for $98.05, and nice new 40-legged bug inside. I plugged the 65802 into my //e, after carefully removing the 65C02 I had just put in a week before. Power on, the drive whirrs, RESET works, hurray!

So far I have spent about six hours exploring the new opcodes. I used the new but yet unreleased version 2.0 of the S-C Macro Assembler, naturally. The literature available up till now has been very sketchy on the details of some of the new opcodes and addressing modes. Anyway, no matter how well the printed word is used, the chip itself will always have the final say, the last word.

Which reminds me that I have already had to correct one mis-understanding (bug?). I was not computing the relative offsets for the 16-bit relative address mode. There are two opcodes which use this mode: BRL, Branch Relative Long; and PER, Push Effective address Relative.

BRL can branch anywhere within a 64K memory, using an offset of 16-bits. Compare this with the other relative branches, which use only an 8-bit offset and can only branch inside a 256-byte space centered around the instruction. BRL's offset ranges from -32768 to +32767.

PER pushes two bytes onto the stack. The two bytes pushed are the high byte and then the low byte of the address calculated by adding the 16-bit offset to the current PC-register. For example,

```
       0800- 62 FD FF    PER $0800
       0803-
```

pushes first $08 and then $00 onto the stack. Voila! Now we really can write position independent code! Using the 16-bit mode, I can PER the address of a data item or table onto the stack, and then PLX (Pull to X-register) that address, and access data by LDA 0,X or the like.

Another favorite pair are the two block move instructions: MVN and MVP. With these I can move any block of memory from 1 byte up to 64K bytes from anywhere to anywhere. With the 65802, anywhere is still limited to the 64K address space, but with the 65816 it can be anywhere in 16 megabytes.

To get full advantage of MVP and MVN, you need to be in the 16-bit mode. You get there in two steps: first you turn on the 65802 mode, as opposed to the 6502-emulation mode; and then you set some status bits which select 16-bit memory references and 16-bit indexing.

You turn on the 65802 mode by clearing the new E-bit in the status register. The E-bit hides behind the Carry bit, and you access it with the XCE (Exchange C and E) instruction.

```
        CLC
        XCE     turns on 65802 mode

        SEC
        XCE     turns on 6502 emulation mode
```

Then REP #$30 turns on the 16-bit mode. REP stands for Reset P-bits. Wherever there are one bits in the immediate value, the corresponding status bits will be cleared. Where there are zero bits in the immediate value, the corresponding status bits will be unaffected. The two bits cleared by REP #$30 are the M- and X-bits. If either of these, or both, are zero, the immediate mode of LDA, LDX, LDY, CMP, ADC, SBC, AND, ORA, and EOR become three byte instructions. For example,

```
        LDA ##$1234
```

loads $1234 into the extended 16-bit A-register. The long A-reg gets a new name or two. The high byte is called the B-register, the low byte is still the A-register, and the pair together are called the C-register.

Okay. Now back to the block movers. Both of the moves require some setting up first. You put the 16-bit address of the source block into the X-register, the destination address in Y, and the move count in C. For example, suppose I want to move the block $0800-$0847 up to $0912:

```
        LDX ##$0800     source
        LDY ##$0912     destination
        LDA ##$0047     # bytes - 1
        MVN 0,0         move it
```

As each byte is moved, X and Y are incremented and A is decremented. After all is complete, A will have $FFFF, X=$0848, and Y=$095A.

MVP, on the other hand, decrements the A-, X- and Y-registers for each byte moved. If the block source and destination overlap, you can use the one which moves in the order that prevents mis-copying.

Those two zeroes after the MVN instruction above are two 8-bit values. In the 65802 they don't mean anything, but in the 65816 they are the high 8-bits of the 24-bit addresses of source and destination. In the 65816, you could copy one entire 64K bank to another with just four instructions! And it only takes 3 cycles per byte moved!

The 65802 plugs directly into the 6502 socket in your Apple //e. It may or may not work in older Apples ... I haven't tried it yet. The 65816 will not plug into any current Apple II, even though it also has forty pins. The extra 8-bits of address are multiplexed on the 8 data lines, and the meaning of the other pins is somewhat changed.

Please don't get the idea that plugging in this new chip will speed up your old software. Old software will stay in the 6502 emulation mode, and will run at exactly the same pace as before. New software can be written which will take advantage of the new features, and it can be a little faster, more compact, and so on. The exciting future of the 65802 and 65816 lies not inside old Apples, but in the Apples yet to be born. I am dreaming of a 4-megahertz, 1- to 8-megabyte Apple ...

Meanwhile, here is a REAL example. Way back in the January 1981 issue of Apple Assembly Line I published a General Move Subroutine. It was set up as a control-Y command for the monitor. As an improvement over the monitor M-command, it could move blocks which overlapped either up or down in memory without repeating the leading bytes.

The following program takes advantage of the MVN and MVP commands to greatly speed up and shrink my previous effort. The old one took 149 bytes, the new one only 80. Disregarding all the setup time, which also improved, the time to move a single byte changed from a minimum of 16 cycles to a consistent 3 cycles.

Lines through 1090 describe how to set up and run the program, but don't even TRY it until you get a 65802 chip into your Apple! The new opcodes will do amazing things in an old 6502 chip, but nothing at all like intended.

Line 1100, the .OP 65816 directive, tells version 2.0 that it should allow and assemble the full 65816 instruction set.

Lines 1180-1250 are executed if you use $300G after assembling, or if you BRUN it from a type-B file.

A1, A2, and A4 are monitor variables which are setup by the control-Y command. When you type, for example, 800<900.957^Y (where by ^Y I mean control-Y), $800 is stored in A4, $900 in A1, and $957 in A2.

Lines 1270-1290 save the three registers, and these will be restored later at lines 1500-1520. Lines 1320-1340 get us unto the 16-bit mode described above. Just before returning to the monitor we will switch back to 6502 emulation mode, at lines 1480-1490.

Lines 1360-1390 calculate the "#bytes-1" to be moved, by using 16-bit subtraction. Note that the opcodes assembled are exactly the same as they would be for 8-bit operations; the cpu does 16-bit steps here because we set the 16-bit mode.

Lines 1410-1460 determine which direction the block is to be moved: up toward higher memory addresses, or down toward lower addresses. By using two separate routines we prevent garbling the move of an overlapping block.

Lines 1610-1660 move a block down. It is as easy as rolling off a log.... Just load up the registers, and do an MVN command.

Lines 1680-1760 move a block up. Here we need the addresses of the ends of the blocks, so lines 1690-1720 calculate the end address for the destination. Then we do the MVP command, and zzaappp! it's done.

```
                        1000 *SAVE S.GENERAL MOVER
                        1010 *--------------------------------
                        1020 *    BRUN the program to set it up as
                        1030 *       a control-Y monitor command.
                        1040 *--------------------------------
                        1050 *    Use like the Monitor M-command:
                        1060 *       A1 -- Source start address
                        1070 *       A2 -- Source end address
                        1080 *       A4 -- Destination start address
                        1090 *--------------------------------
                             1100          .OP 65816
                             1110          .OR $300
                             1120 *--------------------------------
3C-                          1130 A1       .EQ $3C,3D
3E-                          1140 A2       .EQ $3E,3F
42-                          1150 A4       .EQ $42,43
00-                          1160 BLKSIZ   .EQ $00,01
                             1170 *--------------------------------
                             1180 CONTROL.Y.SETUP
000300- A9 4C                1190          LDA #$4C
000302- 8D F8 03             1200          STA $3F8
000305- A9 10                1210          LDA #GENERAL.MOVER
000307- 8D F9 03             1220          STA $3F9
00030A- A9 03                1230          LDA /GENERAL.MOVER
00030C- 8D FA 03             1240          STA $3FA
00030F- 60                   1250          RTS
                             1260 *--------------------------------
                             1270 GENERAL.MOVER
000310- 48                   1280          PHA
000311- 5A                   1290          PHY
000312- DA                   1300          PHX
                             1310 *--------------------------------
000313- 18                   1320          CLC           65816 MODE
000314- FB                   1330          XCE
000315- C2 30                1340          REP #$30      16-BIT MODE
                             1350 *--------------------------------
000317- 38                   1360          SEC           Compute block length - 1
000318- A5 3E                1370          LDA A2
00031A- E5 3C                1380          SBC A1
00031C- 85 00                1390          STA BLKSIZ
                             1400 *--------------------------------
00031E- A5 3C                1410          LDA A1
000320- C5 42                1420          CMP A4        Determine direction of move
000322- 90 05                1430          BCC .1        ...UP
000324- 20 32 03             1440          JSR MOVE.DOWN
000327- 80 03                1450          BRA .2        ...ALWAYS
000329- 20 3C 03             1460 .1       JSR MOVE.UP
                             1470 *--------------------------------
00032C- 38                   1480 .2       SEC           RETURN TO 6502 MODE
00032D- FB                   1490          XCE
00032E- FA                   1500          PLX
00032F- 7A                   1510          PLY
000330- 68                   1520          PLA
000331- 60                   1530          RTS
                             1600 *--------------------------------
```

```
                        1610 MOVE.DOWN
000332- A6 3C           1620        LDX A1         Source start address
000334- A4 42           1630        LDY A4         Destination start address
000336- A5 00           1640        LDA BLKSIZ     # Bytes - 1
000338- 54 00 00        1650        MVN 0,0
00033B- 60              1660        RTS
                        1670 *--------------------------------
                        1680 MOVE.UP
00033C- 18              1690        CLC
00033D- A5 42           1700        LDA A4
00033F- 65 00           1710        ADC BLKSIZ
000341- A8              1720        TAY            Destination end address
000342- A6 3E           1730        LDX A2         Source end address
000344- A5 00           1740        LDA BLKSIZ     # Bytes - 1
000346- 44 00 00        1750        MVP 0,0
000349- 60              1760        RTS
```

Out of Print..............................Bob Sander-Cederlof

After printing the mini-review of Gene Zumchak's "Microprocessor Design and Troubleshooting" last month, we naturally started receiving orders for the book. I had some on order from Sams, but Lo! It is now out-of-print! I talked with someone inside Sams and they said it will probably remain out-of-print.

I talked with the author directly, and I believe that if necessary he will re-publish the book himself. It is a worthy book, and needs to be available. He wants to update some of the material, too. We'll let you know when we can get it again.

You may have noticed that "computer" books are now the "in" thing to publish. I would not be surprised if some publishers began having serious difficulties because of their eagerness to grab this market. They are publishing fluff for the neophytes, forgetting the really useful technical titles. I hope Sams does not forget how it got where it is today.

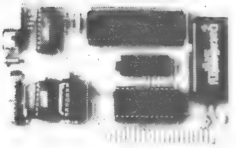
Meanwhile, as Art Carlson says, "If you see a book you need you had better get while it is still available."

On this same subject, let's see if we can put some pressure on Apple to make their reference manuals more readily available. I find that very few (hardly any) Apple dealers will stock or even special order the ProDOS, //e, and //c Reference Manuals. More than twice I have been told that (for example) the //e manual had never been published, even though I bought a copy at a store many moons ago. It seems that Apple will only sell the books in bundles of five or more of the same title, and then only to Apple dealers. Apple dealers seem to not want to order five or more of what are a relatively slow moving item. After all, they are not book stores. And consequently, Apple gets the erroneous impression that they really do not need to publish the manuals, because no one is buying them! If you know anyone in Apple, pass the word to them: WE DO WANT REFERENCE MANUALS. Maybe it does make sense not to ship a copy of every manual with every computer, but some means MUST be available for EVERY owner to buy the manuals he needs.

Apple Peripherals Are All We Make

That's Why We're So Good At It!

THE NEW TIMEMASTER II H.O.



- Absolutely, positively, totally PRO-DOS and DOS 3.3 compatible.
- Time in hours, minutes, seconds and milliseconds (the ONLY PRO-DOS compatible card with millisecond capability).
- 24 hour military format or 12 hour with AM/PM format.
- Date with year, month, day of week and leap year.
- The easiest programming in BASIC.
- Eight software controlled interrupts so you can run two programs at the same time (many examples are included).
- Compatible with ALL of Apple's languages. Many sample programs for machine code, Applesoft, CP/M and Pascal on 2 disks.
- On-board timer lets you time any interval up to 48 days long down to the nearest millisecond.
- Rechargeable nickel-cadmium battery will last over 20 years.
- Two BSR/serial ports for future expansion.

Full emulation of all other clocks. Yes, we emulate Brand A, Brand T, Brand P, Brand C, Brand S and Brand M too. It's easy for the H.O. to emulate other clocks, we just drop off features. That's why the H.O. can emulate others, but none of the others emulate us.
The Timemaster II H.O. will automatically emulate the correct clock card for the software you're using. You can also give the H.O. a simple command to tell it which clock to emulate. This is great for writing programs for those poor unfortunates that bought some other clock card.
Of course, most programs will use the Timemaster II H.O. in its native mode, but its comforting to know that you can use programs written for other products without any modification.

REMOTE CONTROL

Our BSR X-10 interface option for the H.O. allows you to remotely control up to 16 lights and electrical appliances through your BSR X-10 home control system in your home or office. You're already wired because a BSR system sends its signals over regular 120 volt wiring. That means you can control any electrical device in your home or office without any additional wiring.

PRICE $129.00 BSR Option $49.00

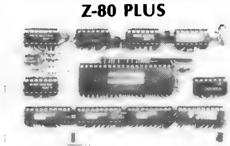
MemoryMaster IIe 128K RAM Card

- Expands your Apple IIe to 192K memory.
- Provides an 80 column text display.
- Compatible with all Apple IIe 80 column and extended 80 column card software (same physical size as Apple's 64K card).
- Can be used as a solid state disk drive to make your programs run up to 20 times FASTER (the 64K configuration will act as half a drive).
- Permits your IIe to use the new double high resolution graphics.
- Automatically expands Visicalc to 95K storage in 80 columns! The 64K config. is all that's needed, 128K can take you even higher.
- PRO-DOS will use the MemoryMaster IIe as a high speed disk drive.
- The 64K MemoryMaster IIe will automatically expand Apple Works to 55K storage. The 128K MemoryMaster IIe will expand Apple Works to 101K storage.
- High Speed disk emulation for BASIC, Pascal and CP/M is available at a very low cost. NOT copy protected.
- Documentation included, we show you how to use all 192K.

If you already have Apple's 64K card just order the MEMORYMASTER IIe with 64K and use the 64K from your old board to give you a full 128K. (The board is fully socketed so you simply plug in more chips.)

MemoryMaster IIe with 128K	**$249**
Upgradeable MemoryMaster IIe with 64K	**$169**
Non-Upgradeable MemoryMaster IIe with 64K	**$149**

Z-80 PLUS



- TOTALLY compatible with ALL CP/M software.
- The only Z-80 card with a special 2K "CP/M detector" chip.
- Fully compatible with microsoft disks (no pre-boot required).
- Specifically designed for high speed operation in the Apple IIe (runs just as fast in the II+ and Franklin).
- Runs WORD STAR, dBASE II, COBOL-80, FORTRAN-80, PEACHTREE and ALL other CP/M software with no pre-boot.
- A semi-custom I.C. and low parts count allows the Z-80 Plus to fly thru CP/M programs at a very low power level. (We use the Z-80A at fast 4MHZ.)
- Does EVERYTHING the other Z-80 boards do, plus Z-80 interrupts.

Don't confuse the Z-80 Plus with crude copies of the microsoft card. The Z-80 Plus employs a much more sophisticated and reliable design. With the Z-80 Plus you can access the largest body of software in existence. Two computers in one and the advantages of both, all at an unbelievably low price.

PRICE $139.00

VIEWMASTER 80

There used to be about a dozen 80 column cards for the Apple, now there's only ONE.

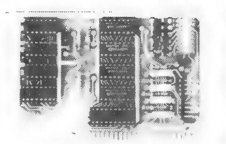
- TOTALLY Videx Compatible.
- 80 characters by 24 lines, with a sharp 7x9 dot matrix.
- On-board 40/80 soft video switch with manual 40 column override.
- Fully compatible with ALL Apple languages and software—there are NO exceptions.
- Low power consumption through the use of CMOS devices.
- All connections are made with standard video connectors.
- Both upper and lower case characters are standard.
- All new design (using a new Microprocessor based C.R.T. controller) for a beautiful razor sharp display.
- The VIEWMASTER incorporates all the features of all other 80 column cards, plus many new improvements.

	PRICE	BUILT IN SOFTSWITCH	SHIFT KEY SUPPORT	LOW POWER DESIGN	80 COLUMN HOME	7x9 DOT MATRIX	LIGHT PEN INPUTS	40 COLUMN OVERRIDE	INVERSE CHARACTERS
VIEWMASTER	**159**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
SUPERTERM	MORE	NO	YES	NO	NO	NO	NO	YES	YES
WIZARD80	MORE	NO	NO	NO	NO	YES	NO	YES	YES
VISION80	MORE	YES	YES	NO	NO	YES	NO	NO	NO
OMNIVISION	MORE	NO	YES	NO	NO	NO	NO	YES	YES
VIEWMAX80	MORE	YES	YES	NO	NO	YES	NO	NO	YES
SMARTERM	MORE	YES	YES	NO	NO	NO	YES	YES	NO
VIDEOTERM	MORE	NO	YES	YES	NO	YES	YES	NO	YES

The VIEWMASTER 80 works with all 80 column applications including CP/M, Pascal, WordStar, Format II, Easywriter, Apple Writer II, VisiCalc, and all others. The VIEWMASTER 80 is THE MOST compatible 80 column card you can buy at ANY price!

Thousands **SOLD** at $179 **NOW ONLY $159.00**

SUPER MUSIC SYNTHESIZER - END MOCKINGBOREDOM



- Complete 16 voice music synthesizer on one card. Just plug it into your Apple, connect the audio cable (supplied) to your stereo, boot the disk supplied and you are ready to input and play songs.
- It's easy to program music with our compose software. You will start right away at inputting your favorite songs. The Hi-Res screen shows what you have entered in standard sheet music format.
- Now with new improved software for the easiest and the fastest music input system available anywhere.
- We give you lots of software. In addition to Compose and Play programs, 2 disks are filled with over 30 songs ready to play.
- Easy to program in Basic to generate complex sound effects. Now your games can have explosions, phaser zaps, train whistles, death cries. You name it, this card can do it.
- Four white noise generators which are great for sound effects.
- Plays music in true stereo as well as true discrete quadraphonic.
- Full control of attack, volume, decay, sustain and release.
- Our card will play notes from 30HZ to beyond human hearing.
- Automatic shutoff on power-up or if reset is pushed.
- Many many more features.

PRICE $159.00

Our boards are far superior to most of the consumer electronics made today. All I.C.'s are in high quality sockets with mil-spec. components used throughout. P.C. boards are glass-epoxy with gold contacts. Made in America to be the best in the world. All products work in the APPLE IIE, II, II+ and Franklin. The MemoryMaster IIe is IIe only. Applied Engineering also manufactures a full line of data acquisition and control products for the Apple; A/D converters and digital I/O cards, etc. Please call for more information. All our products are fully tested with complete documentation and available for immediate delivery. All products are guaranteed with a no hassle **THREE YEAR WARRANTY.**

Texas Residents Add 5% Sales Tax
Add $10.00 If Outside U.S.A.

Send Check or Money Order to:
APPLIED ENGINEERING
P.O. Box 798
Carrollton, TX 75006

Call (214) 492-2027
8 a.m. to 11 p.m. 7 days a week
MasterCard, Visa & C.O.D. Welcome
No extra charge for credit cards

Corrections to Line Number Cross Reference..........Bill Morgan

Allen Miller just called up from Hong Kong (at 3:30 AM his time!) to report a problem with the Line Number Cross Reference program in the August issue. It seems that as published it only prints out the first line number list in each chain. The troublemaker is line 4560, which says BNE .1. Well .1 is the next line, so the routine is always exiting after only one pass. Line 4560 should read BNE PRINT.CHAIN, to go back to the beginning rather than on to the end.

Then Chuck Welman called to point out yet another problem. It seems that an undefined line number greater than the last line of the program caused LCR to head off into the wilderness. When I investigated this one it proceeded to get even stranger. LCR would hang only if the undefined line number was greater than 19668! Less than 19668 came out just right, and equal to 19668 worked, but LCR mistakenly said the line was defined. Now here was a real creepy crawler of a bug!

Well the problem turned out to be in the CHECK.DEFINITION routine. Here are the offending lines:

```
4790 .4      LDY #0
4800         LDA (PNTR),Y     lo-byte of next line address
4810         PHA
4820         INY
4830         LDA (PNTR),Y     and hi-byte
4840         STA PNTR+1
4850         PLA
4860         STA PNTR
4870         JMP CHECK.DEFINITION
```

This code is called when CHECK.DEFINITION wants to get the next line of the Applesoft program. The trouble comes up because there is no check for end-of-program. Sooner or later we come to the zero bytes that mark the end, load up PNTR with zeroes, and go back to CHECK.DEFINITION to try what seems to be the next line. That routine then compares the line number we are checking to the contents of locations 2 and 3 of memory, which Applesoft has loaded with D4 and 4C. Now $4CD4 equals 19668, so that's where that funny number came from!

Here is a slightly rearranged, working version of lines 4790-4870. Note that we have reversed the hi-lo byte sequence and added a check for a zero hi-byte:

```
4790 .4      LDY #1
4800         LDA (PNTR),Y     hi-byte of next line address
4805         BEQ .2           end of program?
4810         PHA
4820         DEY
4830         LDA (PNTR),Y     and lo-byte
4840         STA PNTR
4850         PLA
4860         STA PNTR+1
4870         JMP CHECK.DEFINITION
```

Index to Articles in "Apple Assembly Line", Volume 4

Macintosh Assemblers...............................Lane Hauck

San Diego, CA

I have the privilege* of Beta testing two 68000 assemblers for the Macintosh -- the one from Apple (Workshop), and the one from Mainstay. Mainstay is the "serious" side of Funsoft.

* (If you are masochistic, and enjoy little surprises like alert boxes with no messages or GoAwayButtons in them, frequent crashes, and system fonts abruptly changing; you too might want to become a Beta Tester.)

I've gotten permission from both Apple and Mainstay to talk about these products. The versions I'm testing are preliminary, and therefore subject to change.

The Workshop is in "version 0.6" release, and is expected to be available about October (I'd guess November). The Mainstay product is scheduled for early October release, and judging from their staff and working hours, I think they'll make it. (I visited them in Agoura, CA, and found a very smart and hard working group of programmers.)

Although both assemblers do the same thing -- translate 68000 source programs into runnable programs on the Macintosh -- they couldn't be more different in how they operate!

The Apple Assembler

The Workshop has several parts. EDIT, ASM, LINK and EXEC are four applications that do the actual code development. Additionally, RMAKER creates resource files from text source files created by EDIT. And finally, MacDB and its associated "Nub" programs provide debug support for when your code doesn't run.

The development system can run on one drive, but two are highly recommended.

EDIT: This is a DISK BASED editor, so the short document frustrations of MacWrite are avoided. Additionally, you can open up to four documents, and cut and paste between them (a la Lisa)! This is a bare bones (but wonderful) editor, without fancy fonts or formatting. One improvement over the Lisa editor: it has a "reverse tab" -- hitting backspace from a tab stop takes you back not one space, but back one tab position. This is a great convenience when you're entering formatted source code.

ASM: Supports conditional assembly, macros (both "Lisa-type" and new "Mac-type"). It's tailored to the Mac development environment (for example it helps you write relocatable code).

Toolbox support is provided by special, compressed equate files (they are compressed by a program called PacSyms, which you can use to compress your own equate files). The Workshop provides all the Trap and symbol equates mentioned in Inside Macintosh.

LINK: Links ".REL" code modules produced by the Assembler, and (eventually -- not working yet) output files from RMAKER to produce a final file, complete with your code and resources. Takes its direction from a ".LINK" text file.

EXEC: Lets you automate the entire ASM-LINK process. One great improvement over the Lisa version: you can direct EXEC to reenter the editor if any assembly or link errors occur.

FIVE debuggers are supplied. MacDB is the best, most visible debugger I'ver ever seen. It requires two Macs (or one Mac and a Lisa running MacWorks). The Workshop will be supplied with an interconnect cable for two Macs. Other debugger versions (which don't require the second Mac) let you debug from an 8-line onscreen window on the Mac, and from any remote terminal.

This is a professional, complete, "industrial strength" 68000 assembly language development package. Its utilization of the Macintosh environment is total and outstanding. My only real quibble is that it takes a fair amount of time (a few minutes) to "turn" one cycle from EDIT to running the new code. A hard disk would presumably improve this greatly.

If you're an "interactive" programmer who likes to make changes and see their results QUICKLY, you might be interested in the Mainstay Assembler.

The Mainstay Assembler

If you've ever used any of the assemblers for the Apple II from S-C Software, you'll feel right at home with the Mainstay environment. It's patterned after the S-C 68000 Cross Assembler, and it looks and feels just like you're running on an Apple //e!

The fact that none of the Macintosh interface is used will bother some, especially the Mac purists. Mainstay's intention is to get a quality assembler to market quickly, and the approach they've taken allows this to happen. I don't mind non-fidelity to the Mac interface in a DEVELOPMENT product -- we developers are EXPECTED to put up with all sorts of indignities!

This is an absolute assembler, meaning that your code module is produced with an address origin, and it is loaded and run at that address. It does not produce "linkable" code modules, as does the Apple Workshop Assembler. In fact no linker is supplied or required.

The Editor is built in, and it functions much like the Apple II. The cursor is moved around with keyboard commands. The Editor has BASIC-like line numbers and the normal complement of line-number oriented commands (RENumber, COPY, MOVE, etc.).

Resources are handled right inside your source code (remember there is only one code "module"). This is more convenient than the Apple "RMAKER" approach.

The Assembler supports conditional assembly, macros, and local labes. It takes a novel approach in how it is installed and run on the Macintosh.

When you start the Assembler, it grabs a large chunk of memory from the application heap, and uses it for storing the symbol table, source code, and object code. Typing MEM shows you exactly where these three memory areas are. While you're in the Assembler environment, your code "stays put", so you can deal with absolute addresses without fear that the memory manager will move things around on you.

This means that you can edit, assemble, and test your code IMMEDIATELY, without goin through a linking and (optionally) a resource compiling step. This is the primary strength of this assembler -- it allows "quicklook" programming which is ideal for experimentation and learning the Macintosh system.

Eventually you will want to make your application an "installable" Macintosh program, so you should get into the habit of writing position independent code. The Mainstay package will supply the tools necessary to make your application runnable on the Mac. It will also contain Toolbox and Operating System equate files.

There are some nice "Apple II-like" features, such as typing DIR to look at the disk catalog. In the Mac environment, you have to exit the application and get back to the desktop to see your files. You can also type "EJECT" and eject a disk immediately. I like to do this just before running new code, to protect disks from my runaway test programs that mysteriously fire up the disk drive.

Having this assembler, a Mac, and a copy of INSIDE MACINTOSH might just be the most efficient way to learn the Macintosh. The prime benefit of this assembler is its very high speed in moving between editing, assembling, and running your test code.

Which One?

Which assembler would I recommend? At this stage I'd have to give the universal Computer Salesman answer: "It depends."

The Apple one allows you to write separate code modules, assemble them, and then link them together later. This allows you to utilize already written and debugged modules in new programs.

Another advantage of the "linker" approach is that a single module can be changed and reassembled, and then linked to other already-debugged modules. This saves reassembling the whole shebang every time you make a change.

If you like this "relocatable assembler" approach, you'll want the Apple Assembler. (If you're comfortable with the Lisa Assembler, ditto.)

The Mainstay Assembler, by contrast, is an absolute assembler - it puts code at a particular place in memory (set by an ORG - origin statement), and allows only one "module" -- your entire program. (Better write relocatable code if you want it to run as an application, though!)

The Mainstay Assembler is so fast (especially if you put a "LIST OFF" directive at the beginning of your code), that it negates the speed advantage of the linked module approach. I would guess that it takes you from source code edit to running reassembled code in about one-twentieth the time required by the Apple Assembler. if you're an "interactive" programmer who likes to see results of program changes FAST, the Mainstay Assembler is for you.

If time is a factor, the Mainstay product will ship within a week; the Apple Assembler is supposed to come out in October, but I doubt it.

If you're unhappy with "non-Mac-user-interface" products, you're better off with the Apple version. The operation of the Mainstay assembler is a bit strange at first, but anyone with Apple II roots will adjust quickly.

Here's a factor I consider very important: Apple is a "Pascal house" with almost no support given to assembly language programming of the Macintosh. I've found their support in this area dismal.

The Mainstay Assembler is a major committment by this small company. I've had quite a bit of technical interaction with them, and have found them to be very intelligent, motivated, and responsive. I've had indications that you'll be able to expect not only Assembler support from Mainstay, but also some Macintosh support as well.

[10/15 -- The folks at Mainstay tell me they started shipping last week, so we should have some copies for sale by the time you read this. The introductory price is $100. -- Bill]

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).